IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR U.S. LETTERS PATENT

Title:

PROCESSOR AND PROCESSOR METHOD OF OPERATION

Inventor:

Blaine D. Gaither
1600 Serramonte Drive
Fort Collins, Colorado 80524-1718
Citizenship: United States

PROCESSOR AND PROCESSOR METHOD OF OPERATION

BACKGROUND

[0001] At the present time, although a wide variety of processor-types are commercially available, most processors utilize a relatively common architecture. For example, FIGURE 1 depicts a block diagram of processor 100 implemented utilizing a known architecture. Processor 100 includes memory interface unit 101 that interfaces via a bus with lower level memory (e.g., a lower level cache or main memory) which is not shown. Processor 100 further includes data cache 102 and instruction cache 103. Data cache 102 and instruction cache 103 typically employ a relatively efficient implementation of random access memory (RAM), such as synchronous RAM (SRAM), for on-chip memory transactions. In other known architectures, data cache 102 and instruction cache 103 are integrated within a single element of processor 100.

[0002] Processor 100 further includes translation lookaside buffer (TLB) 104. TLB 104 typically provides a mapping between virtual addresses and physical addresses. Load/store unit (LSU) 105 with load queue 106 and store queue 107 generates the virtual addresses associated with loads and stores. LSU 105 further accesses data cache 102. Pre-fetch and dispatch unit (PDU) 108 accesses instruction cache 103 to fetch instructions for provision to functional units 110. Instruction buffer 109 may be utilized to facilitate pre-fetching of instructions before the instructions are required by the instruction pipeline. Functional units 110 provide processing functionality, such as integer operations, floating point operations, and/or the like, for various instructions supported by processor 100.

[0003] The architecture of processor 100 utilizes two techniques to optimize processor performance. The first of these techniques is caching. Data cache 102 and instruction cache 103 increase system performance because there is a higher probability that, once processor 100 has accessed a data element at a particular address, a subsequent access to an address within the same general memory region will occur. Accordingly, when processor 100 requests data from a particular address, data (e.g., a cache line) from a plurality of "nearby" addresses is transferred from a slower, main memory to cache memory. Then, when processor 100 requests data from another address within the cache line, there is no need to access the slower main memory. The data from the other address may be obtained directly from cache memory.

[0004] The second technique involves implementing functional units 110 in a manner that enables functional units 110 to execute multiple instructions simultaneously. In general, an instruction starts execution if the respective issue conditions are satisfied. If not, the instruction is stalled until the conditions are satisfied. Such interlock (pipeline) delay typically causes the interruption of the fetching of successive instructions or equivalently necessitates "no-op" instructions. One important limitation upon the ability to execute multiple instructions simultaneously is the availability of data to be processed by the execution of the instructions. For example, if an instruction causes the multiplication of two floating point numbers, the instruction cannot be executed until the two floating point numbers are obtained from memory. Thus, the unavailability of data due to memory constraints can limit the parallel processing capability of a processor.

SUMMARY

[0005] In one embodiment, the present invention is directed to a processor that comprises an instruction pipeline for executing processor instructions wherein the processor instructions define a memory access size and a cache memory for storing cache lines in a plurality of memory banks that have a block size that is greater than the memory access size, the cache memory including mapping logic for storing contiguous groups of bits, of size equal to the memory access size, in different ones of the plurality of memory banks.

BRIEF DESCRIPTION OF THE DRAWINGS

[0006] FIGURE 1 depicts a block diagram of a processor according to the prior art.

[0007] FIGURE 2 depicts a block diagram of a processor according to a representative embodiment.

[0008] FIGURE 3 depicts mapping of a cache line to memory banks and retrieval of data within the cache line during pipeline execution of instructions according to a representative embodiment.

[0009] FIGURE 4 depicts a flowchart for storing data in memory banks of cache memory according to a representative embodiment.

[0010] FIGURE 5 depicts a flowchart for retrieving data from memory banks of cache memory according to a representative embodiment.

[0011] FIGURE 6 depicts a flowchart for a method of operation of a processor.

DETAILED DESCRIPTION

[0012] Some representative embodiments of the invention are directed to systems and methods for minimizing bank conflicts when accessing cache memory. Specifically, when known caching mechanisms store cache lines in memory banks and processors retrieve a number of bits of data from the memory banks that is less than the block width of the memory banks, memory bank contention may occur between sequential instructions. That is, a first instruction to a first memory address may initiate memory access to a given memory bank. A second instruction to a second memory address that is immediately after the first memory address may then initiate memory access to the same memory bank. If the first memory access is still pending, the second instruction is stalled and cannot begin until the first memory access is completed. Thus, known caching mechanisms reduce the parallel processing capacity of a processor.

[0013] In contrast, certain representative embodiments of the invention enable conflicting memory accesses to be minimized by suitably configuring the manner in which data is stored within the banks of cache memory according to the defined memory access size of respective instructions. Such representative embodiments are particularly advantageous for execution of scientific or other computationally intensive programs. Specifically, programs that sequentially access data from memory may be executed with a greater degree of parallelism according to these embodiments than possible when executed according to known processor architectures.

[0014] FIGURE 2 depicts processor 200 according to one representative embodiment of the invention. Processor 200 includes memory interface unit 101. Memory interface unit 101 facilitates the storage and retrieval of cache lines from main memory or a lower level cache in response to data cache 201 and instruction cache 103. Specifically, when a cache line is to be retrieved, data cache 201 communicates with memory interface unit 101. Memory interface unit 101 utilizes a suitable bus (not shown) to obtain the data associated with the cache line from

main memory. Memory interface unit 101 communicates the data to data cache 101. Mapping logic 202 receives the data from memory interface unit 101 and stores the data across, for example, banks 203-1 through 203-4 according to a defined mapping scheme. It shall be appreciated the additional or fewer memory banks 203 may be utilized to implement various representative embodiments.

[0015] In this representative embodiment, the mapping scheme is selected according to the size of the memory bank blocks and the size of the memory access size associated with one or more instructions. The block size of memory banks 203 is selected to be greater than the memory access size associated with one or more instructions. Mapping logic 202 stores a cache line across memory banks 203 by dividing a cache line into groups of bits where the size of the groups is related to the memory access size. Specifically, the groups of bits are stored in memory banks 203 by mapping logic 202 such that no two contiguous groups of bits are stored in the same memory bank 203. By doing so, sequential accesses to addresses in a cache line according to the memory access size will not stall due to memory bank contention.

[0016] FIGURE 3 depicts mapping of cache line 301 to memory banks 203-1 through 203-4 and retrieval of data within cache line 301 during pipeline execution of instructions according to this representative embodiment. For the purpose of discussing this representative embodiment, it will be assumed that the cache line size is eight words (which have 16 bits each) and that the respective memory access instructions retrieve a single word (16 bits). Also, it will be assumed that the width of memory banks 203 is two words (32 bits). A mapping scheme for implementation by mapping logic 202 could map the first and fifth words of cache line 301 to bank 203-1, the second and sixth words to bank 203-2, the third and seventh words to bank 203-3, and the fourth and eighth words to bank 203-4. It shall be appreciated that this mapping scheme is by way of example only. In other embodiments, other mapping schemes are employed to store adjacent words or adjacent groups of bits in respective memory banks 203.

[0017] To illustrate the relationship between the mapping scheme and the execution of sequential codes, a hypothetical application that sums the values in a vector will be discussed. In the hypothetical application, the vector is represented by a data structure stored in consecutive memory locations. The magnitude of each vector element is represented by a single word (16 bits). The application performs various operations that cause one or several cache lines

containing the data structure to be loaded into data cache 201. The hypothetical application iteratively retrieves a respective word from a memory address offset from the starting address of the vector data structure and adds the retrieved word to a register.

[0018] In the iterative loop, the same instructions are repetitively obtained by PDU 108 (shown previously in FIGURE 2) and enter instruction pipeline 303 to be executed by one or several of functional units 110 (shown previously in FIGURE 2). The first memory access instruction 304-1 in the iterative loop enters pipeline 303 first. Instruction 304-1 accesses the word associated with the first vector element which is assumed to be the first address (i.e., the first word) in cache line 301. Instruction pipeline 303 causes the retrieval of the first word from memory bank 203-1 to be initiated. Due to memory bank limitations, the memory access may require more than one cycle to complete the retrieval of the relevant word.

[0019] Instruction 304-2 that accesses the word associated with the second vector element enters instruction pipeline 303 almost immediately after instruction 304-1. Due to the sequential nature of the code of the hypothetical application, instruction 304-2 causes a memory access to the memory address (word two in cache line 301) immediately after the word addressed by instruction 304-1. However, this memory address is not located sequentially in memory banks 203 after the prior memory address. Specifically, the fifth word is stored sequentially after word 1. The memory address associated with the second word is mapped to another memory bank 203 (i.e., memory bank 203-2). Thus, instruction pipeline 303 causes the retrieval of the second word from memory bank 203-2 to be initiated while memory bank 203-1 is still performing the retrieval of the first word.

[0020] This iterative process continues by initiating the retrieval of words three and four from memory banks 203-3 and 203-4 by instructions 304-3 and 304-4. When the iterative process reaches instruction 304-5, another memory access to memory bank 203-1 occurs. However, by the time instruction 304-5 enters instruction pipeline 303, a sufficient number of instructions have been executed to allow the previous memory access associated with memory bank 203-1 to be completed. Thus, the latency associated with memory banks 203-1 through 203-4 does not stall the instruction pipeline when mapping logic 202 is suitably implemented.

[0021] FIGURE 4 depicts a flowchart for storing data in memory banks of cache memory according to one representative embodiment. In step 401, a cache line is retrieved from

lower level memory which may be main memory or a lower level cache. In step 402, the retrieved cache line is divided into segments that are related to the memory access size of one or more processor instructions. In step 403, the segments are stored in memory banks of the cache memory such that the memory banks contain more than one segment and consecutive segments of the cache line are stored in different memory banks.

[0022] FIGURE 5 depicts a flowchart for retrieving data from memory banks of cache memory according to some representative embodiments. In step 501, an instruction accessing a first memory location is received in an instruction pipeline. In step 502, the retrieval of data from the first memory location from a first memory bank is initiated. In step 503, an instruction accessing a second memory location that is immediately after the first memory location is received in the instruction pipeline. In step 504, the retrieval of data from the second memory location is initiated while the first retrieval transaction is still pending.

[0023] FIGURE 6 depicts a flowchart for a method of operation of a processor. The method comprises, in step 601, executing instructions in an instruction pipeline wherein the instructions define a memory access size. The method further comprises, in step 602, storing cache lines in memory banks of cache memory included in the processor, the memory banks having a block size that is greater than the memory access size, wherein the storing causes contiguous groups of bits, of size equal to the memory access size, to be stored in different ones of the memory banks.

[0024] Some representative embodiments of the invention may provide a number of advantages. Specifically, the performance of cache memory accesses may be improved by avoiding pipeline delays associated with conflicting memory bank access. By doing so, these representative embodiments enable sequential processing of data to occur in an efficient manner. Thus, applications that perform scientific or other computationally intensive tasks may be executed with a greater degree of parallelism that possible utilizing known processor designs.